

Code optimizations for efficient embedded systems

Peter Marwedel
University of Dortmund + ICD
Dortmund, Germany

The computational requirements of embedded systems increase at a rapid rate. Reasons for this trend include the increased use of multimedia and the need to use more efficient coding techniques for wireless communications. For example, consider the difference in the computational requirements of 2 G and 3 G mobile phones. Due to this trend, the limited resources of embedded systems become a serious bottleneck. It has been recognized that the available energy may be the single most limiting factor. The increasing importance of software is a second trend in embedded systems. Looking at the combined effect of the two trends, it can be concluded that embedded software must be efficient. Therefore, this tutorial focuses on techniques for improving the efficiency of software. We will start the tutorial with a motivation for the focus on software optimizations.

We will then address problems resulting from the increasing speed gap between processors and memories (this problem is also known as the “memory wall” which eventually will be hit). Approaches for bridging this gap which are in use in PCs (caches) cannot guarantee the predictable real-time behaviour required by most embedded systems. New memory hierarchy architectures have been proposed as a remedy to this problem. In particular, these hierarchies involve scratch pad memories (SPMs). SPMs are small (and therefore fast and energy efficient) memories which are mapped into the memory address space. SPMs (called tightly coupled memories by ARM) are available with many processors, but currently the tool support is very limited. We show that both the energy consumption as well as the computed worst case execution time (WCET) can be reduced significantly by code optimizations exploiting the memory architecture. We present algorithms using single loading of memory objects as well as compiler-controlled paging-like reloading of the SPM. We also present algorithms including context-saving and restoring for the SPM in a multiprogramming environment. We demonstrate how designers can take advantage of the new optimizations without replacing existing tool chains.

We will then present source code techniques that can be applied at a high level, prior to using a compiler. Techniques that include intra- and inter-array folding, floating-point to fixed point conversion, and loop transformations such as loop merging, loop splitting, loop permutation, loop tiling, and intra- and inter-array folding. A new transformation called loop body splitting will also be demonstrated. Finally, we will present optimization techniques that can be used in a compiler or as a postpass-optimization. These include address assignment techniques which exploit auto increment- and decrement-addressing modes. We will briefly show the effect of multimedia- or short-vector instructions and how they can be used to improve the software efficiency.

The tutorial is designed such that researchers as well as industrial attendees should benefit from the presentation of the material.